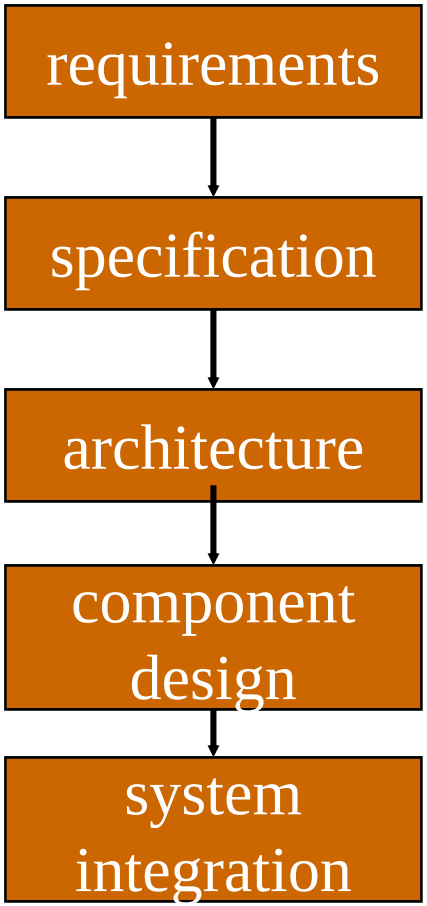
CH4-a
Software Architecture for Embedded System

課程：嵌入式系統與軟體工程
開發學校：中央大學資工系
陳慶瀚

Hierarchical Design Flow



Top-down vs. bottom-up

- Top-down design:
  - start from most abstract description;
  - work to most detailed.
- Bottom-up design:
  - work from small components to big system.
- Real design uses both techniques.

Stepwise refinement

- At each level of abstraction, we must:
  - analyze the design to determine characteristics of the current state of the design;
  - refine the design to add detail.

Requirements

- Plain language description of what the user wants and expects to get.
- May be developed in several ways:
  - talking directly to customers;
  - talking to marketing representatives;
  - providing prototypes to users for comment.

Functional vs. non-functional requirements

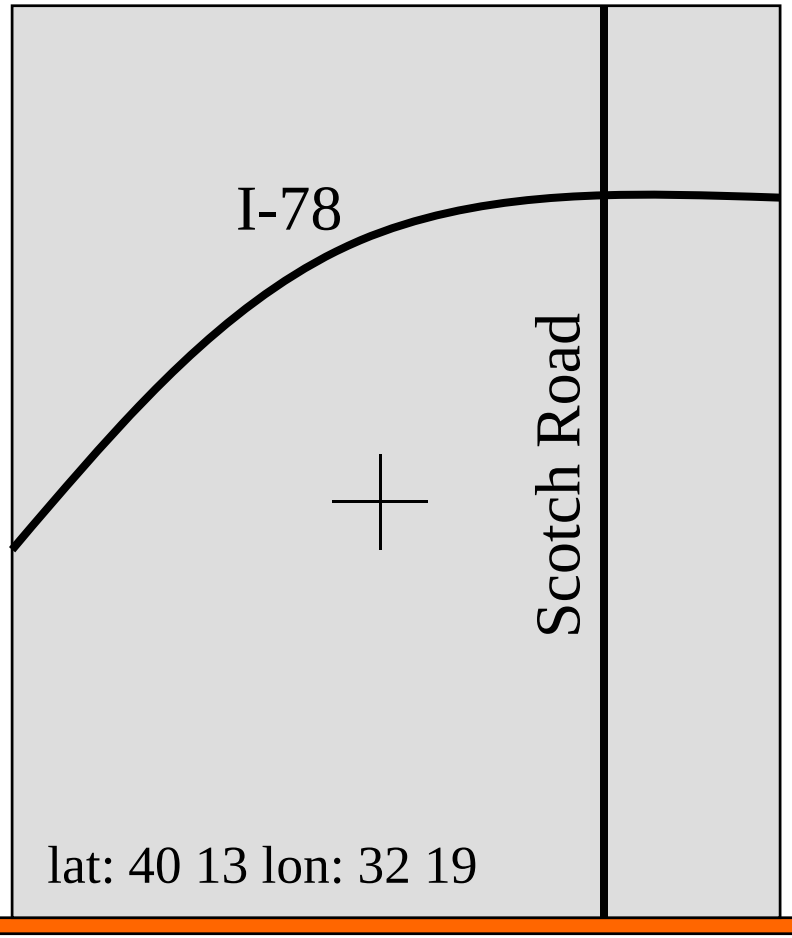
- Functional requirements:
  - output as a function of input.
- Non-functional requirements:
  - time required to compute output;
  - size, weight, etc.;
  - power consumption;
  - reliability;
  - etc.

Requirements form

name
purpose
inputs
outputs
functions
performance
manufacturing cost
power
physical size/weight

Example: GPS moving map requirements

- Moving map obtains position from GPS, paints map from local database.



GPS moving map needs

- Functionality: For automotive use. Show major roads and landmarks.
- User interface: At least 400 x 600 pixel screen. Three buttons max. Pop-up menu.
- Performance: Map should scroll smoothly. No more than 1 sec power-up. Lock onto GPS within 15 seconds.
- Cost: $100 cost of goods sold.

GPS moving map needs, cont′d.

- Physical size/weight: Should fit in hand.
- Power consumption: Should run for 8 hours on four AA batteries.

GPS moving map requirements form

name	GPS moving map
purpose	consumer-grade moving map for driving
inputs	power button, two control buttons
outputs	back-lit LCD 400 X 600
functions	5-receiver GPS; three resolutions; displays current lat/lon
performance	updates screen within 0.25 sec of movement
manufacturing cost	$100 cost-of-goods-sold
power	100 mW
physical size/weight	no more than 2: X 6:, 12 oz.

Specification

- A more precise description of the system:
  - should not imply a particular architecture;
  - provides input to the architecture design process.
- May include functional and non-functional elements.
- May be executable or may be in mathematical form for proofs.

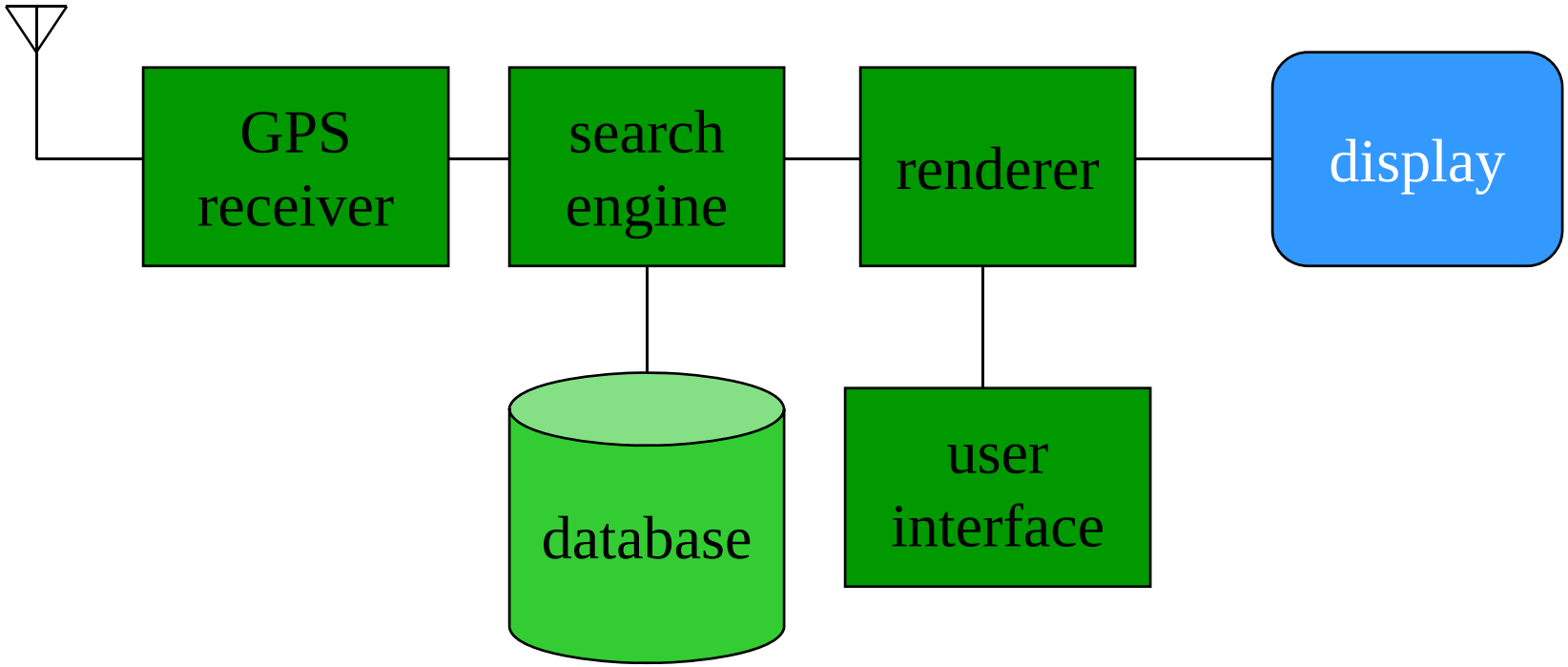
GPS specification

- Should include:
  - What is received from GPS;
  - map data;
  - user interface;
  - operations required to satisfy user requests;
  - background operations needed to keep the system running.

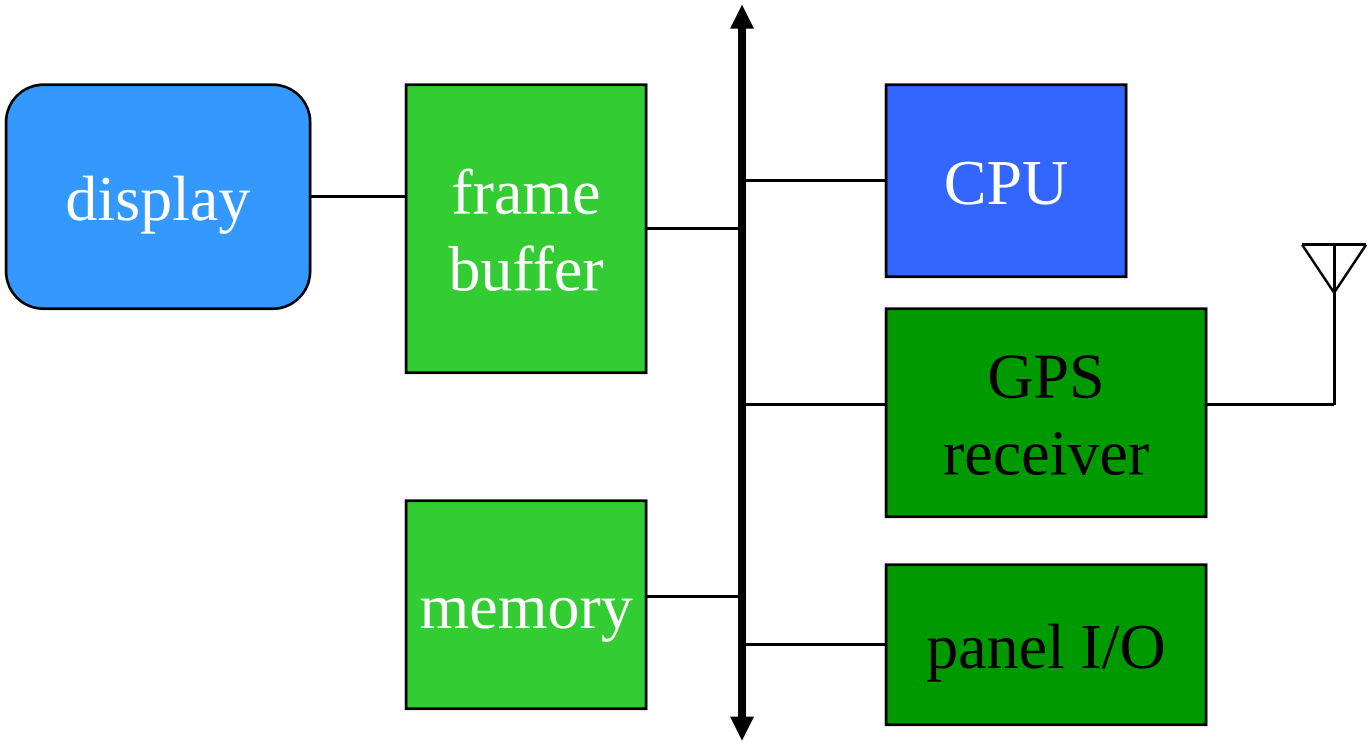
Architecture design

- What major components go satisfying the specification?
- Hardware components:
  - CPUs, peripherals, etc.
- Software components:
  - major programs and their operations.
- Must take into account functional and non-functional specifications.

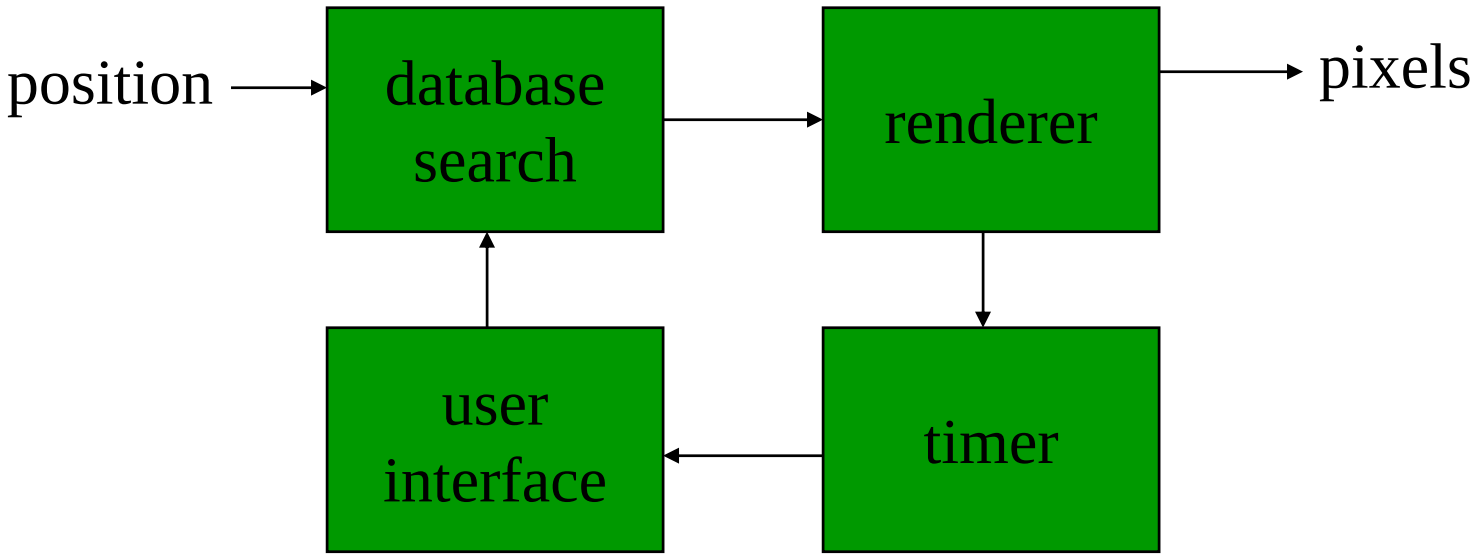
GPS moving map block diagram



GPS moving map hardware architecture



GPS moving map software architecture



Designing hardware and software components

- Must spend time architecting the system before you start coding.
- Some components are ready-made, some can be modified from existing designs, others must be designed from scratch.

System integration

- Put together the components.
  - Many bugs appear only at this stage.
- Have a plan for integrating components to uncover bugs quickly, test as much functionality as early as possible.

Embedded Programming Concept

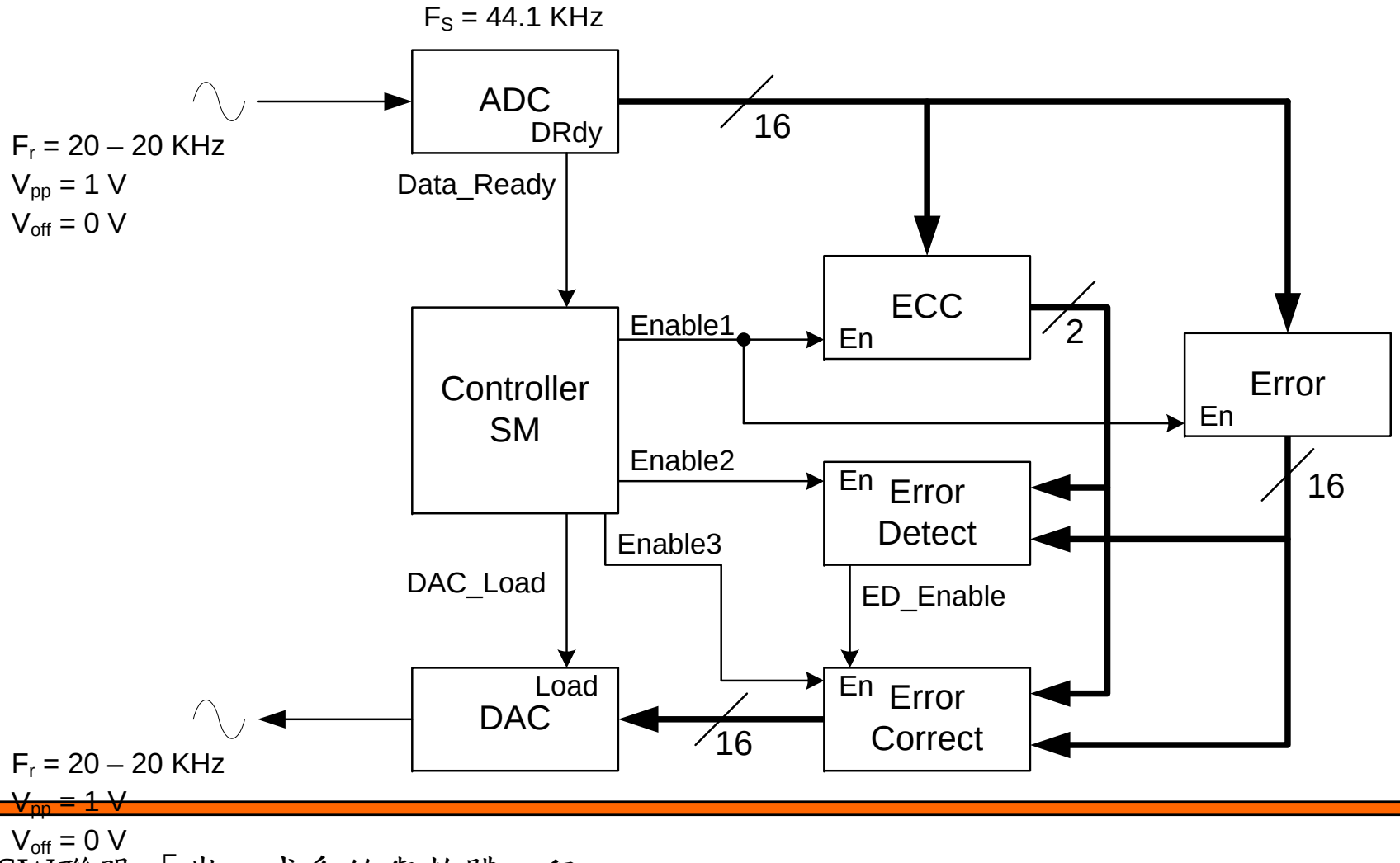
Execution Flow

- A diagram that describes the major steps in the sequential software flow.
- Illustrates each logical step and decision point in the overall design of the software.
- Provides a starting point for the design of the software.
- Is to embedded software as logical architecture is to hardware.
- Is mapped later to Routines, Functions or Objects

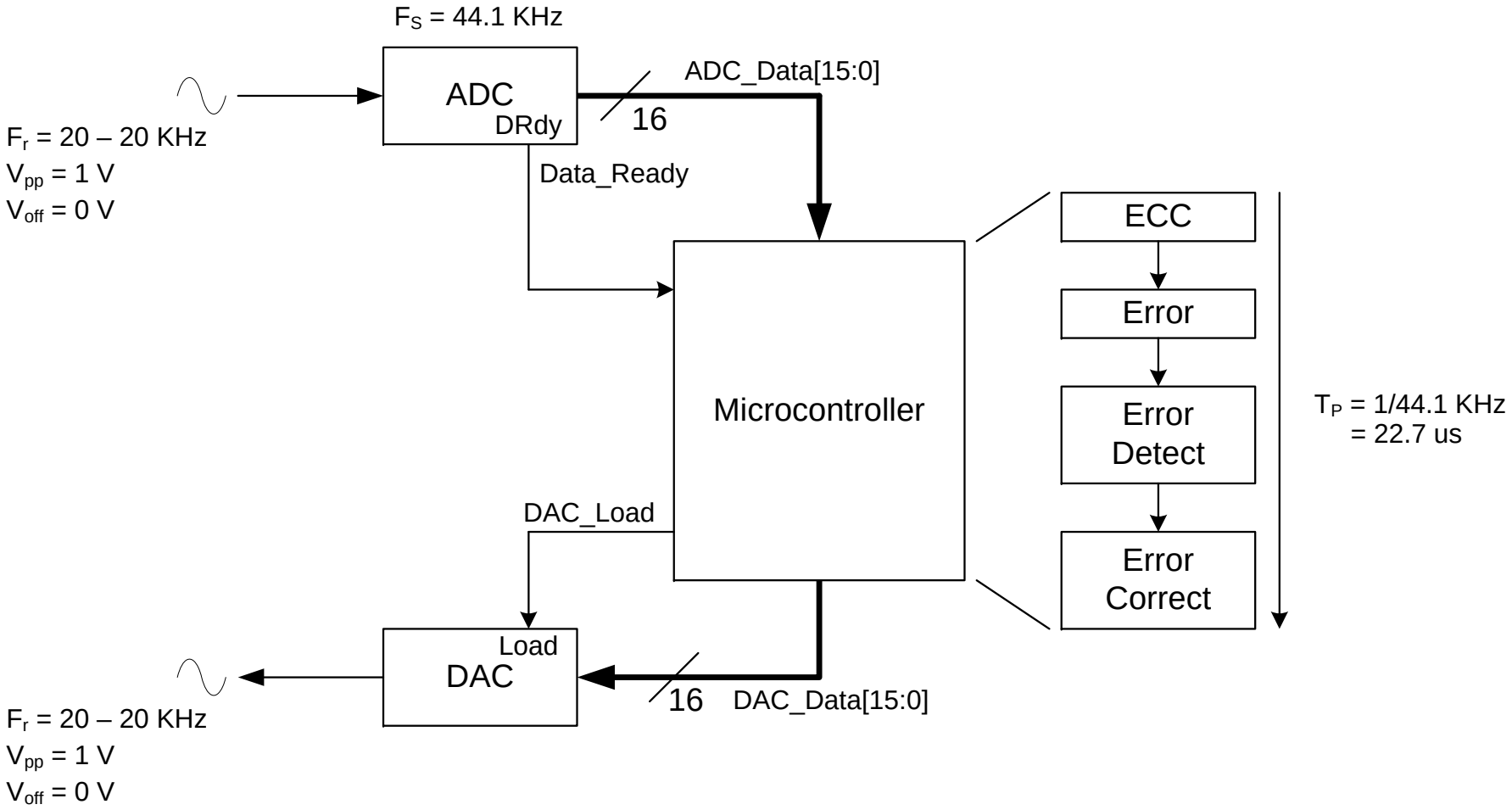
Before Designing Execution Flow

1. Identify which logical architectural modules are mapped to software.
2. Identify what time period all logical software function must be completed in.
3. Identify any additional overhead functionality due to modules mapped to software and/or characteristics of the physical design. ie. Polling/Interrupts

Logical Design Architecture Example



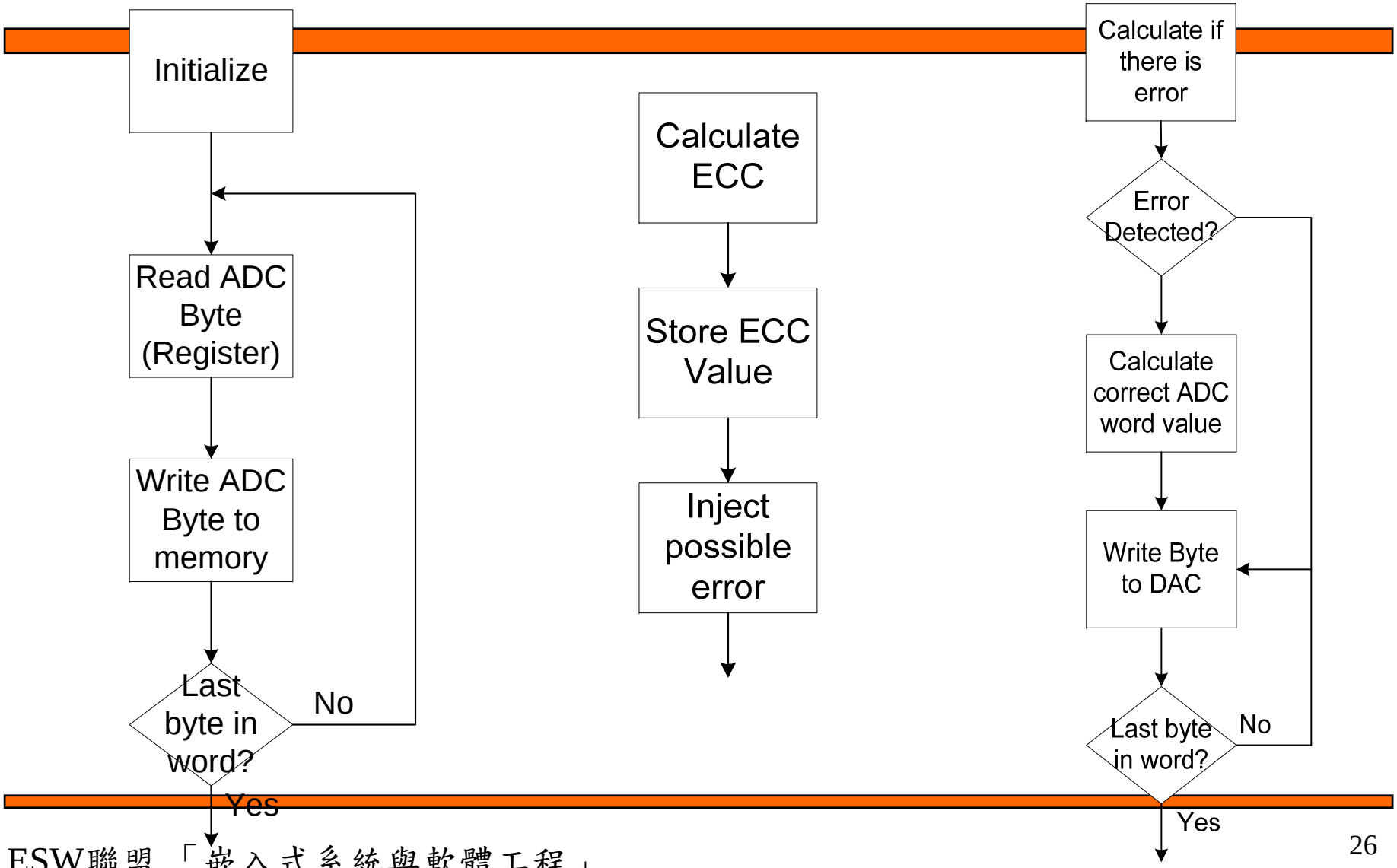
Physical Architecture Mapping



Role of Microcontroller

- Read 16-bit word from ADC
- Calculate Error Correction Code (2-bit)
- Inject possible error
- Detect if there is an error
- If there is an error correct it
- Write new 16-bit word to DAC

Execution Flow Diagram



Data Throughput in SW

- What is synchronous data flow?
  - Data arrives at regular intervals
- What is asynchronous data flow?
  - Data does not arrive at regular intervals
- What is isochronous data flow?
  - Data must be delivered within certain time constraints

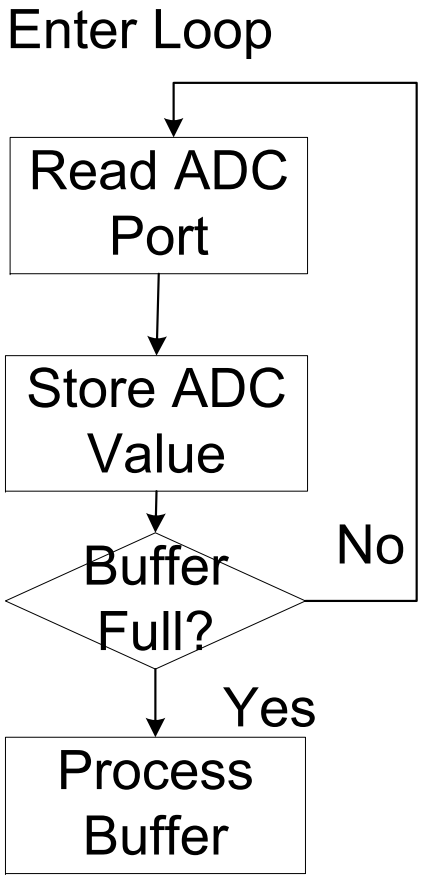
Data Flow Practices

- Polling
- Interrupt triggered (blocking)
- Interrupt triggered (non-blocking)
- Event driven
  - Often referred to as interrupt driven

Sample Problem

- Need to receive 16 bytes into a buffer and then process the buffer
- Bytes arrive asynchronously

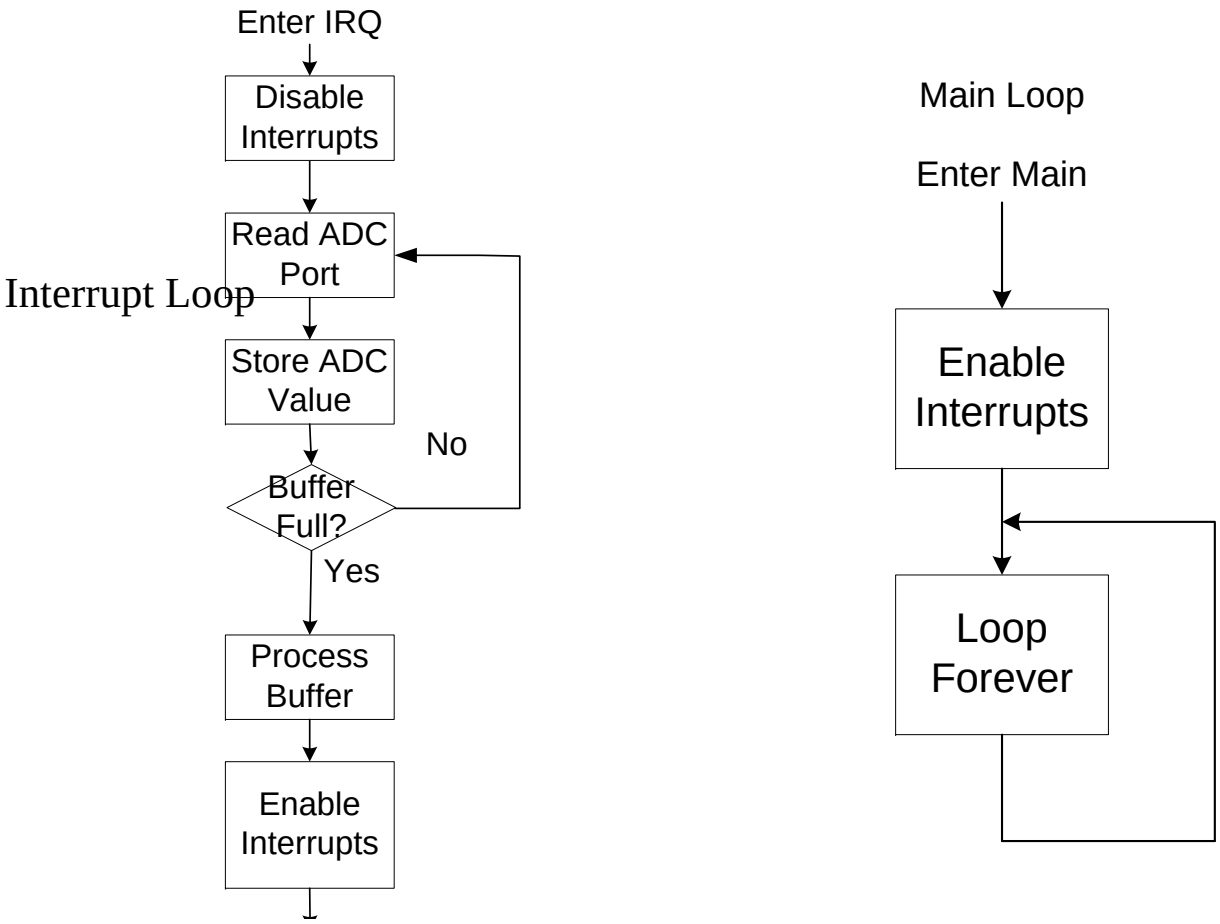
Polling Overview



Polling

```
main
  do forever
    count = 0
    while count < 16
      while byte not ready
        nop
      get byte
      buffer[count] = byte
      incr count
    process buffer
```

Interrupt (Blocking) Overview

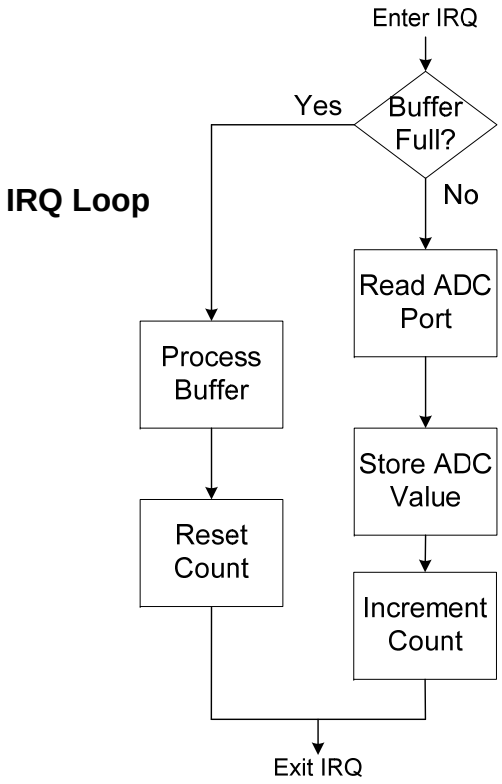


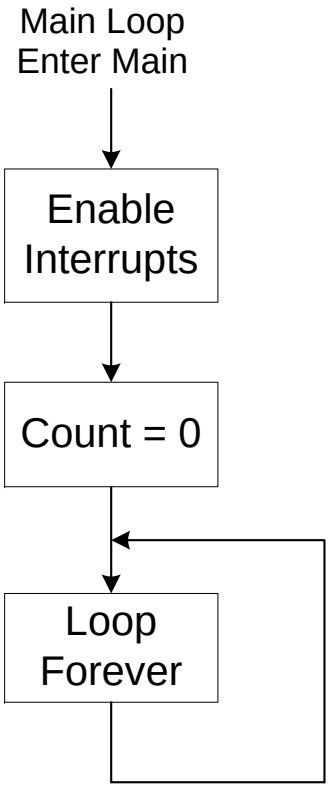
Interrupt Triggered (Blocking)

```
interrupt rx_byte
  disable interrupts
  count = 0
  while count < 16
    get byte
    buffer[count] = byte
    incr count
  process buffer
  enable interrupts
  return
```

```
main
  enable interrupts
  do forever
    nop
```

Interrupt (Non-Blocking) Overview



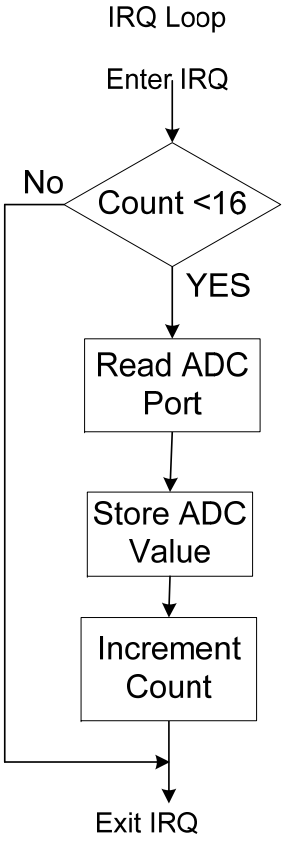


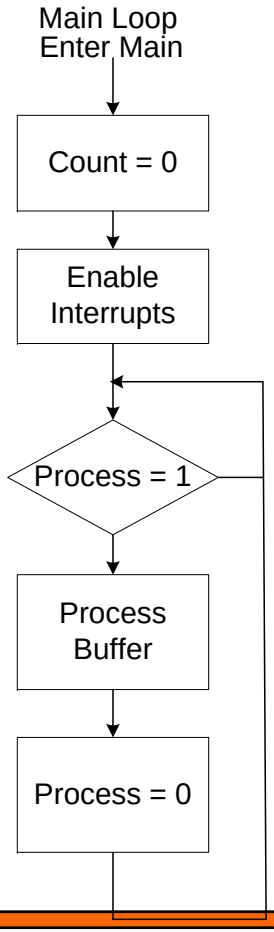
Interrupt Triggered (Non-blocking)

```
interrupt rx_byte
  if count < 16
    get byte
    buffer[count] = byte
    incr count
  else if count = 16
    process buffer
    count = 0
  return
```

```
main
  count = 0
  enable interrupts
  do forever
    nop
```

Event Driven Overview





Event Driven

```
interrupt rx_byte
  if count < 16
    get byte
    buffer[count] = byte
    incr count
  return
```

```
main
  count = 0
  enable interrupts
  do forever
    if count = 16
      process buffer
      count = 0
```

Software Design Overview

- Requirements Analysis
- Architecture
- Design
- Implementation
- Module Testing
- Design Validation Testing

Example Problem

- I want to build a Heads Up Display for my car.
- I would like to see both my engine RPM and fuel economy on my windshield.
- My car has a serial diagnostic interface that provides this data.

Requirements

- System shall have a Heads Up Display (HUD)
- System shall interface with vehicle's onboard diagnostic system
- HUD shall display current RPM
- HUD shall display current fuel economy (MPG)

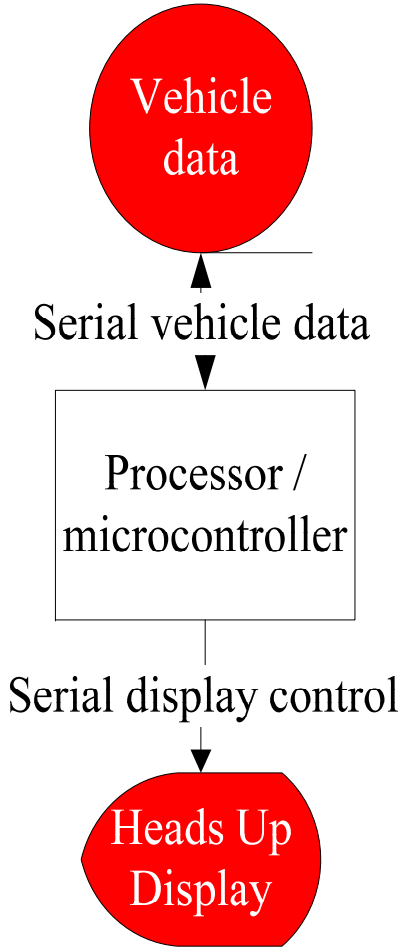
Hardware Constraints

- Vehicle's onboard diagnostic system
  - Needs a wake-up signal sent every 1 second
  - Operates at 10,500 bps
- Heads Up Display
  - 256x128 pixels
  - Full display must be calculated and mirrored
  - Operates at 115,200 bps

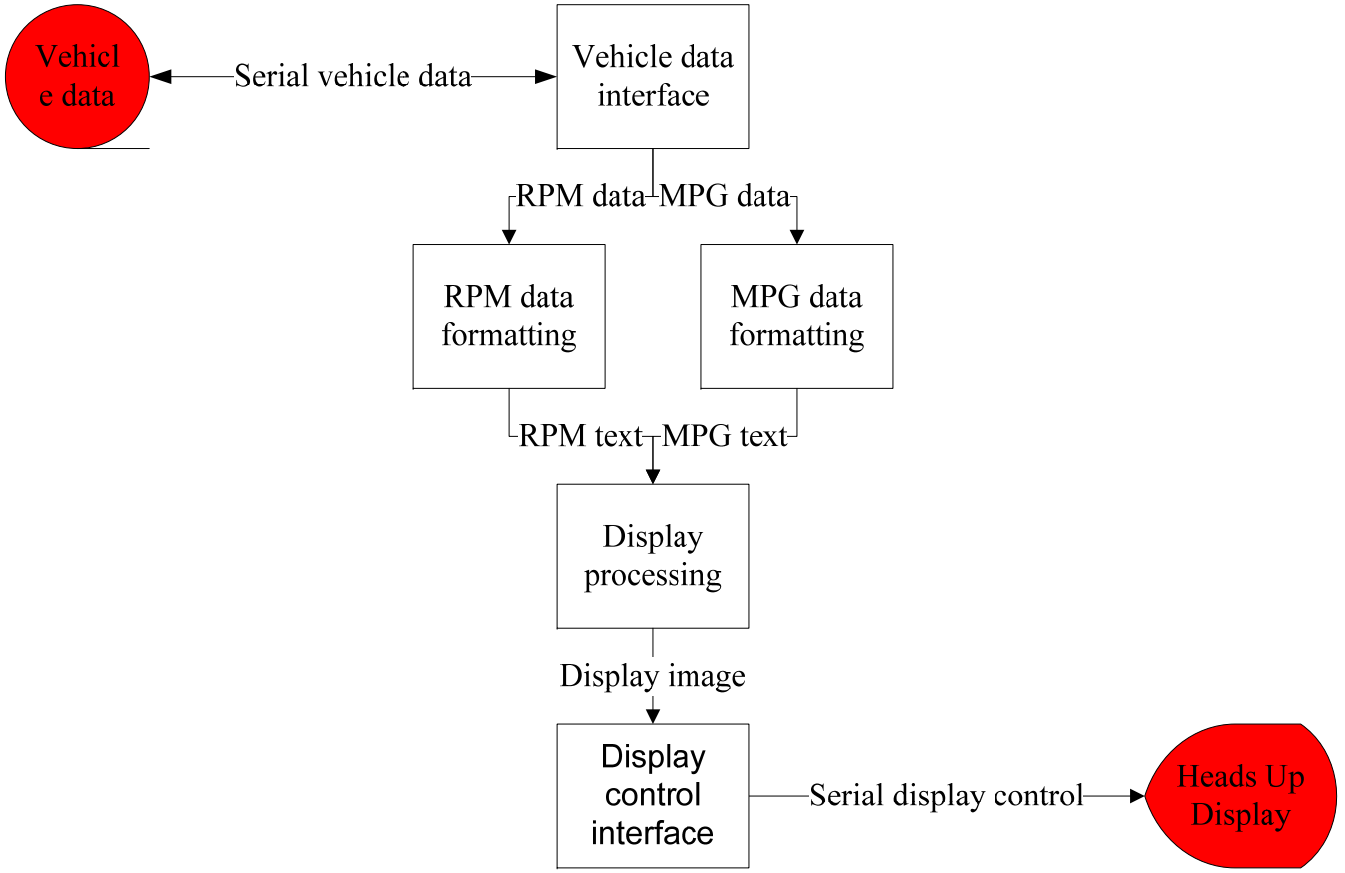
Processor Requirements

- Processor shall have 2 UARTs
  1. Vehicle’s onboard diagnostic system
  2. Heads Up Display
- Processor shall have a timer
  - Vehicle’s onboard diagnostic system wake-up
- Processor shall have more than 8192 bytes of memory
  - Processed display image (4096 bytes)
  - Mirrored display image (4096 bytes)

Hardware Design



Software Architecture



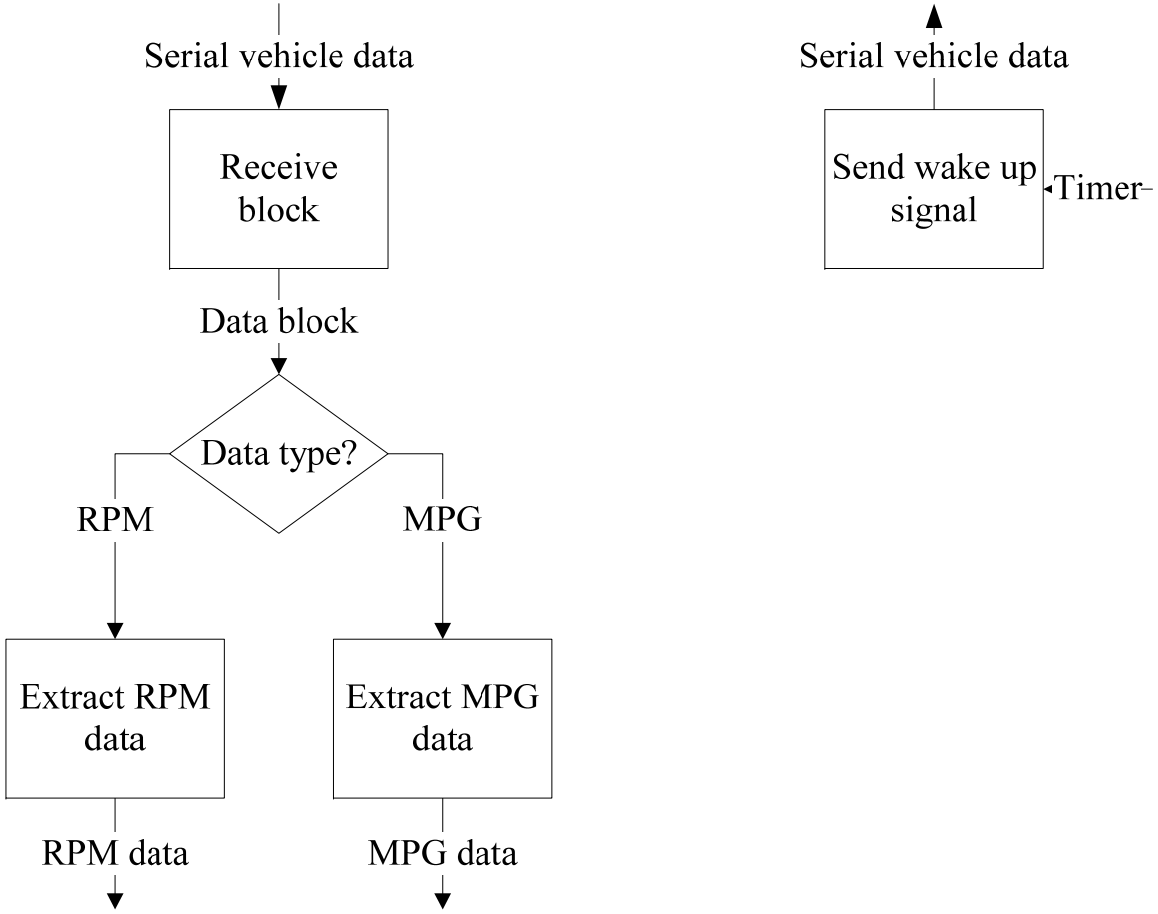
Software Design

- Modules
  - Vehicle Diagnostic Interface (VDI)
  - RPM Data Formatting (RDF)
  - MPG Data Formatting (MPG)
  - Display Processing (DP)
  - Display Control Interface (DCI)
- Main/Initialization

Software Design

- Modules
  - Vehicle Diagnostic Interface (VDI)
  - RPM Data Formatting (RDF)
  - MPG Data Formatting (MPG)
  - Display Processing (DP)
  - Display Control Interface (DCI)

Vehicle Diagnostic Interface

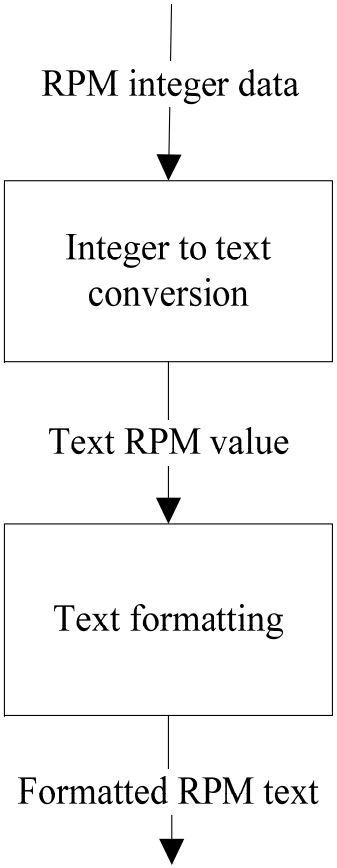


Vehicle Diagnostic Interface

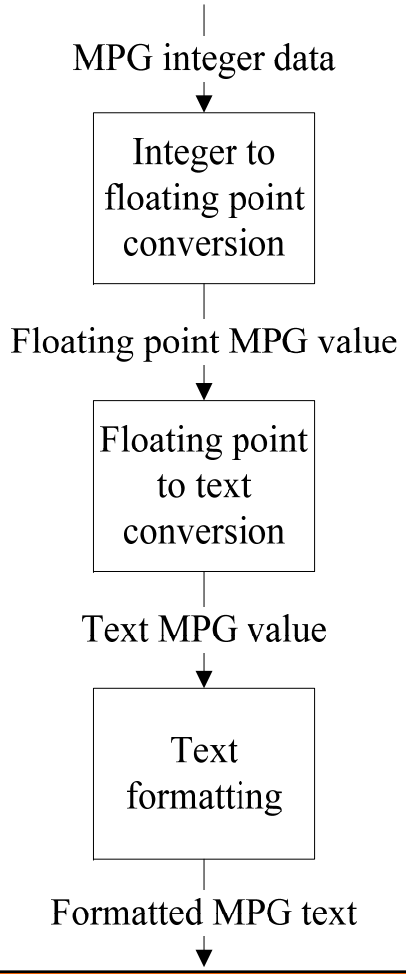
```
interrupt timer
  wakeup = 1
  return

tx_wakeup
  send wakeup control block
  return
```

```
main
  …
  do forever
    …
    if wakeup = 1
      tx_wakeup
      wakeup = 0
    …
```

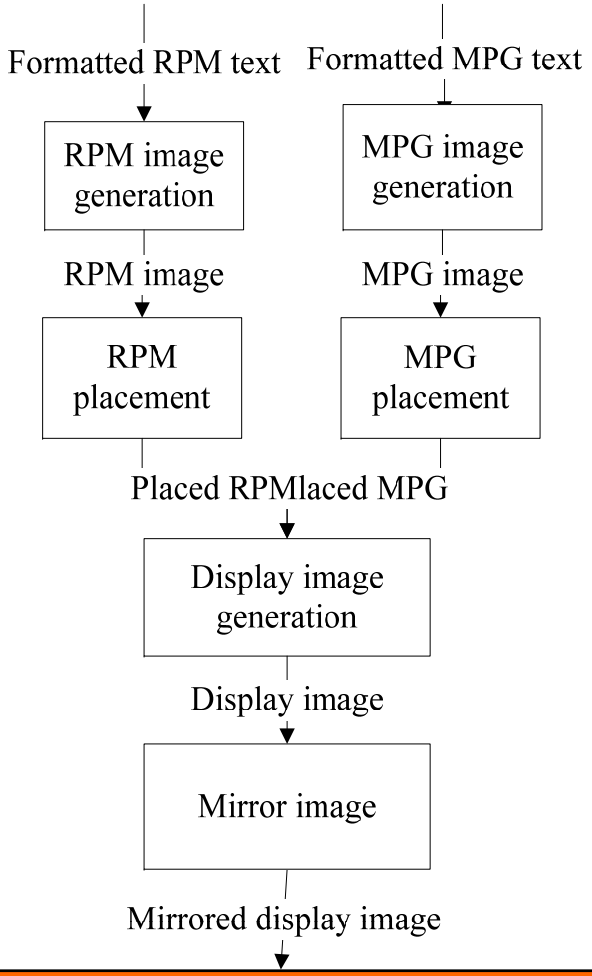
RPM Data Formatting



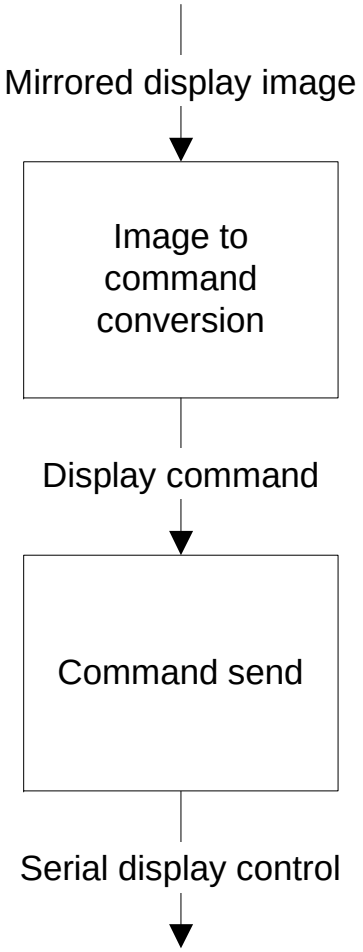
MPG Data Formatting



Display Processing



Display Control Interface



Test Plan

1. Test functionality of every module
2. Test functionality of every module interaction
3. Test functionality of the final system

Implementation

This is when you actually write your code.

Module Testing

- Simple implementations to test a single module or module interaction
- Total testing code will often be larger than the actual system's code base
- Is this good or bad?

Design Verification Testing

- A scripted test plan that guides a tester through use of the system
- A table with the following:
  - Every system requirement
  - Whether or not the requirement was met